Coordination of Distributed Collaborative Activities for Disaster Management

Jörn Franke, François Charoy

Changes done to Coordination of Distributed Collaborative Activities for Disaster Management

Jörn Franke and François Charoy*

October 19, 2012

The paper has been extensively rewritten and revised regarding the style and the wording. Figures that were removed in the WETICE paper due to space constraints have been included and commented, in order to ease the understanding of the paper.

The most important change is the part related to evaluation (VII) at the end of the paper. This part has not yet been published. It describes our proposition regarding the evaluation of the approach that we propose and explains its outcomes and limits.

*J. Franke and F. Charoy is with LORIA-Inria-CNRS, Université de Lorraine, BP 239-54506 Vandoeuvre-lès-Nancy Cedex, France charoy@loria.fr

Coordination of Distributed Collaborative Activities for Disaster Management

Jörn Franke and François Charoy

Abstract— **It is very challenging for different organizations to coordinate together in dynamic situations like a disaster response. Each organization is autonomous and considers the situation from their point of view. There is no central authority to coordinate all operations. To coordinate their actions, organizations need to exchange information on what they are doing. However, they cannot share everything with everybody due to privacy, regulatory or strategic reasons. Currently, they only use e-mail, telephone or fax to exchange information. Thus, it is very difficult for them to detect and handle differences on their perception of the situation. We propose an approach for inter-organizational process management suited to these dynamic scenarios. It allows different organizations to share selected activities by replicating them optimistically in each other workspaces. The underlying system propagate the state changes to all workspace eventually. We explain detecting and handling of two different types of conflicts that can occur in this setting. We provide an implementation and explain how we have derived a first evaluation of the system.**

I. Introduction

According to Gartner and McKinsey, the management of activities in dynamic distributed processes gets more and more importance [1]. We consider processes that take place in the "real world" involving humans belonging to different organizations. In this context, We argue that not only processes must be flexible, but they should enable autonomous organizations to coordinate their activities. Disaster response management domain provides us with critical scenario where coordination counts and where goals shift during an event - dynamic distributed processes. Disaster manager cannot design a structured fully-specified version of these processes due to their dynamic nature. From an inter-organizational perspective, we also claim that it is impossible to create a global shared process that one entity can control. Each organization coordinates its own activities based on its experience and governance rules. To coordinate with each other, these organizations must share information about their activity and their reliance on activities of other organizations. When people coordinate with communication tools (e.g. email, phone or fax), they have difficulties to establish an adequate situation overview.

Here, we show that it is possible to design an activity management system to address this problem. We assume that there is no central coordination, but a network of organizations that need to synchronize their actions. We do not know the network structure in advance. Organizations need to exchange information about what they plan, do or have done. They may also send orders to others. This requires to share information about activities and to permit to change it concurrently. We will show how we can detect and handle the conflicts that occur in these cases. Our work is based on an approach presented in [2]. This paper takes into account the inter-organizational dimension, where privacy is important. For example, the police cannot share information about crime investigations with the fire brigade. The main contribution of this paper is to show how it is possible to detect and handle conflicts caused by cooperating organizations. Additionally, we discuss evaluation of systems as the one we present it in this paper.

In the next section, we describe a use case in the field of disaster response management where stakeholders have to coordinate in a distributed fashion. We have worked with end users to develop it [3]. We explain how the activities and their dependencies are modeled as the basis for coordination (based on the framework proposed in [2]) in section three. We describe then how organizations can use the model to coordinate activities in section four. We focus on resolving conflicts when sharing activities between different organizations. We describe the architecture in section five and the implementation in section six. In section seven and eight, we discuss end user feedback and we explain how we can design experiments to evaluate systems as the one we have developed here. Finally, we present related work and give an outlook on future research.

II. Use Case

First, we present a use case that establishes the need for flexibility and coordination among independent organizations. We have derived it from a disaster response use case developed with end users like fire fighters and police officers during the SoKNOS project [3]. These organizations must to work together, but none of them is hierarchical superior to the other. They form an organizational response network. In the simplified version of the use case, three organizations respond to a flood, the police, the fire brigade and the military (figure 1). The military have to protect a chemistry plant from being flooded. They fill sandbags, transport sandbags and build a dam to achieve this objective. The fire fighters are building a dam to protect a residential area from this flood. They rely on the military to provide sandbags to them. The police has to evacuate the residential area if the flood arrives. They have to determine people, warn them, transport them or order shelter. They execute these actions based on the success or failure of the other organization actions. Police activities depend on the success of the dam construction.

J. Franke and F. Charoy is with LORIA-Inria-CNRS, Université de Lorraine, BP 239-54506 Vandoeuvre-lès-Nancy Cedex, France charoy@loria.fr

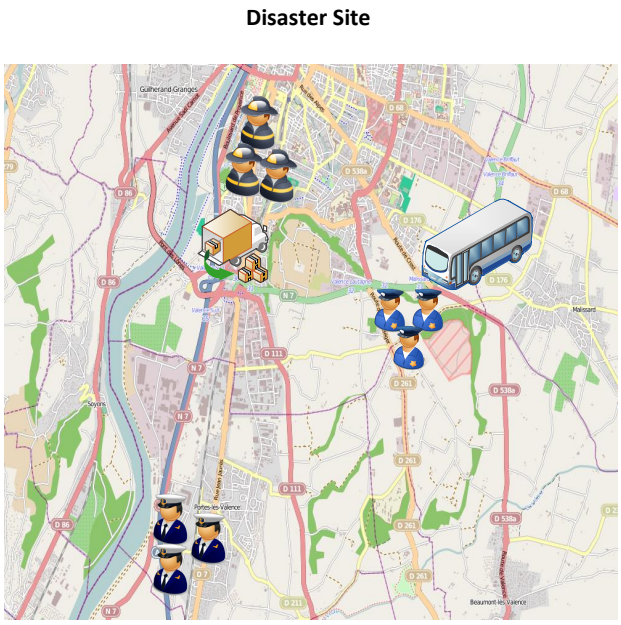

Fig. 1. Scenario

The fire brigade relies on the delivery of sandbags by the military. It is beneficial for each organization to know what is happening and what may concern the other. Of course, each organization needs also to keep some of their actions private (i.e. everything cannot be shared) due to privacy reasons or internal policies.

Anyone can join or leave the organizational network at any time: different regions or states may provide additional command centers for supporting the coordination among different disaster sites. In this case actions may have an impact on each other (building dams at two different places for instance). During our research within the SoKNOS project and interactions with end users (e.g. workshops) we found out that current means, such as email, telephone or fax, cause some problems for coordination. It is almost impossible to get an accurate overview of the status of all ongoing actions. Some organizations think that something is happening while it has failed (an order to close an airport has been given and is assumed to be completed while it is not). It can be very difficult to detect these conflicting views using the traditional means. This leads to confusion about the current situation.

We argue that a process based approach to manage this coordination can address these challenges, even when organizations are autonomous and coordinate the situation from their point of view. The absence of central entity defining how to do the coordination in detail for all organizations can be overcomed.

III. A Framework for Coordination of Activities

In this section, we describe briefly how users model activities and temporal dependencies to describe explicitly coordination. It is based on the framework explained in [2]

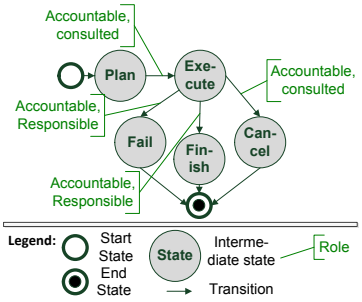

Fig. 2. Example for an activity type with governance roles

that describes also how it is possible to verify and execute its model. We will reuse this ability in the following sections.

Definition 1 An *activity type* $at_d = (S, st, se, f, G)$ represents the management lifecycle of an activity. S is a finite set of activity states; $st \in S$ describes the start state of an activity type; $se \in S$ describes the end state of an activity type (a state without outgoing transition); $st \neq se$ a start state is not an end state; $f : S \rightarrow S$ is a transition function defining the possible transitions from one state to another for one activity type. We can extend the specification of the activity type with governance rules $G = \{g_1, .., g_n\}$. They describe who can transition from one state to another, e.g. $g_x \subseteq f$ is the transition function of the role "x".

Fig. 2 illustrates an example of an activity type. The white circle describes the start state and the black circle describes an end state. Other states are "Plan", "Execute", "Idle", "Fail", "Cancel" and "Finish". We do not allow strongly connected components (i.e. cycles) in the activity type - this causes confusion (cf. [2]). For example, it would be possible to go from state "Execute" to "Fail" and vice versa. Users may have difficulties to understand it particularly, when the status is shared with other users.

Definition 2 An *activity* is defined as $a_i = (uid, name, cat, cs, P)$ where uid is a unique identifier of the activity; $name$ describes the activity; cat is the activity type of the activity; cs is the current state of the activity. The first current state of an activity is the start state st of its activity type. P is the set of participants assigned by the creator of an activity to a governance role in $cat.G$. An activity can change its state in parallel to other activities without affecting them. However, users can establish dependencies between activities, if they perceive it as important. Any further data can be attached to the activity.

Definition 3 A *temporal dependency* is defined as $d_i = (a_s, s_s, a_d, s_d, type)$ with a_s is the source activity; s_s is the state of the source activity; a_d is the destination activity, s_d is the state of the destination activity and $type$ is the type of temporal dependency.

We use Allen's proposed time interval relationships for describing different types of temporal dependencies [4]. Figure 3 illustrates seven of them and omits six inverse dependencies. The dependency changes its state to "Violate" or "Neutral" depending on the order of the state changes

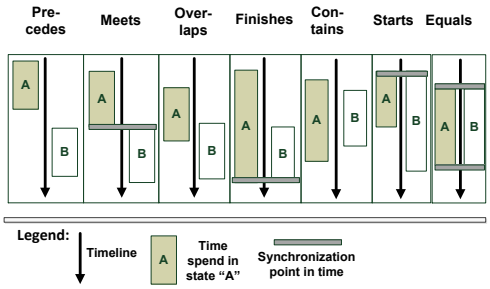

Fig. 3. Types of supported temporal dependencies

of the associated activities (cf. [2]). The relation "overlaps" between the states "Execute" of activity "A" and "B" provides us with an example of a dependency. It indicates that activity "A" has to enter state "Execute" before the activity "B" can enter state "Execute". Activity "A" has to leave state "Execute" before activity "B" does. If it fails to happen in this order then the dependency is violated. This may occur because the situation requires it or because people are not aware of the dependency. In this case, the system can warn the user of the violation. He can then take appropriate actions, such as communicating with the stakeholders of the activities or by creating new ones.

In the subsequent sections, we will provide examples demonstrating how organizations can use this model to coordinate activities. We explain the nature of conflicts that may occur in this distributed setting, how a system can detect and manage them.

IV. Coordination and Conflict on the Inter-Organizational Level

As we explained in the use case section, organizations need to share information about their actions to help their coordination. As we understood it from interviews with disaster management experts, errors come frequently from misunderstanding or lack of information about what the other teams are doing. In this section, we will describe how they can coordinate by sharing activities in a way that is supported by the framework that we propose. We place us in a situation where every organization has access to this framework. We assume that each organization maintains an activity workspace (AW) containing all its internal activities and dependencies. A person from one organization can decide to share an activity with a person of another organization. The selection of organizations and of who shares what with whom is based on an existing social network. For instance, the fire chief knows the police commander. It is out of the scope of this paper.

The person of the other organization can then decide to insert this activity in its AW. This preserves autonomy of both organizations. The shared activity is then replicated in the AWs of both organizations. Users can manage it like any other activity of the workspace. They can create new dependencies from and to this activity. They can change the

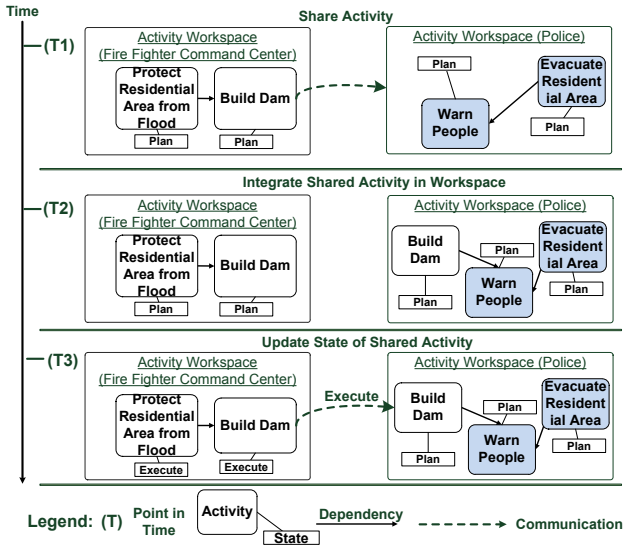

Fig. 4. Example for sharing of activities, integration of shared activities in an activity workspace and updating the state of shared activities

status of this activity in both AWs. The system propagates them optimistically. Thus, both organizations will maintain a partially shared view on the current operations. Optimistic replication means that concurrent state changes can occur that may conflict or have different outcomes regarding dependencies violation. In the remainder of this section, we detail the general principles of the approach and our proposal to deal with conflicts.

A. Sharing of Activities

Sharing of activities enables to coordinate them between different organizations. Our approach of sharing allows organizations to keep their autonomy - sharing is voluntary. They may decide to share activities or not as well as to take into account an activity shared by another organization. We came to this approach based on our interactions with end users in the SoKNOS project.

Sharing can take place between people of different organization, but also between people of the same organization on different levels of the hierarchy. New organisation can join or leave the sharing network at any time.

In our approach, participants model activities and dependencies on an activity workspace (AW). They share some activities with other participants of another AW. Then, they establish dependencies between shared activities and their own activities. In Fig. 4 we provide an example showing the sharing of activities. In the first step (T1), the fire fighter commander shares the activity "Build Dam" with the police commander. In a second step (T2), the police commander has integrated the shared activity "Build Dam" in his AW and created a dependency from the shared activity to his activity "Warn People".

Each workspace maintains a list with all the workspaces where the activity is replicated.

B. Updating States of Shared Activities

We describe in this section how state changes of shared activities are propagated to all AWs, where the activity is replicated.

We do this optimistically. We propagate the state change and detect as well as handle conflicts afterwards. This allows coordinating in an instant like with traditional means. A pessimistic approach would mean to lock the activity for a period of time in which no state changes can be entered by the user. This would limit unnecessarily the possible interaction with the system (cf. also [5]). A pessimistic approach can lead to inaction, because people have to wait until they can provide input or receive input to do action. This is contrary to what happens in disaster response management. It is not appropriate in our case. In Fig. 4, we show in step three (T3) that the fire fighter commander changed the activity "Build Dam" to state "Execute". Since the activity "Build Dam" is shared with the police commander, the change is propagated to the AW of the police commander.

We presented in [6] a protocol for optimistically propagating state changes. The underlying assumption is that messages arrive eventually. The outcome of the protocol is that state changes are applied in any AW where the shared activity is replicated. Applying a state change in a model means detecting if dependencies are violated by it. Since the protocol only provides optimistic replication, we need to detect and handle conflicts afterwards.

In the next two subsections we describe how we can detect and handle two different types of conflicts that can occur after optimistic propagation by this protocol.

C. Detecting and Handling Conflicts with Shared Activity

The first type of conflict occurs when two users change concurrently the state of a shared activity with two different values. For example, when considering the activity type in Fig. 2, a conflict can occur if one person sets an activity based on this activity type to state "Cancel" and the other one to state "Fail" concurrently. This type of conflict is illustrated in Fig. 5. The commander in the command center changes to state "Cancel" the activity "Build Dam" and the commander in the field changes it to state "Fail" in the third step (T3). If we apply the protocol above, the conflict can be detected based on the activity type and the history of state changes of the activity. Indeed, it is impossible to transit from state "Execute" to "Fail" and at the same time from "Execute" to "Cancel" - thus, there is a conflict.

Definition 4 *Conflicting state change history*: Let $\sigma_y = (s_1 \rightarrow s_2, .., s_{n-1} \rightarrow s_n)$ be the execution history with the state changes $s_1 \rightarrow s_2, .., s_{n-1} \rightarrow s_n$ of activity y based on the activity type at. A conflict occurs when: $\exists((s_i \rightarrow s_j) \wedge (s_i \rightarrow s_k) \wedge (s_j \neq s_k)), i = 1..n-1$. This definition means that there is a conflict in the history of state changes if there are two or more state changes originating from the same state (s_i) of the same activity.

As mentioned, we assume that all AWs sharing the activity have eventually the same elements in their execution history. We cannot go twice through the same state without causing conflicts in the history with the activity type. This is only possible when the activity type has cycles, which we excluded by definition.

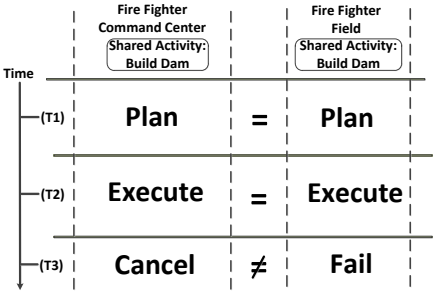

Fig. 5. Example for detecting conflicts caused by state changes of one shared activity "Build Dam"

However, if the user handles this conflict manually, we cannot guarantee that it will be resolved eventually. Thus, we propose an automated approach (cf. for details [7]). Our approach is inspired from [8], but we adapted it to our context, where we do not have a central authority. It uses the governance roles that the creator of an activity can define to resolve automatically the conflict. For example, the commander in the command center has shared the activity "Build Dam" with the commander in the field and both perform conflicting state changes. The commander in the command center changes the state to "Cancel" and the commander in the field to "Fail". Since the commander in the command center has the "accountable" role for the activity, he is higher in the role hierarchy than the commander in the field who is only "responsible". The final state in both workspaces will be "Cancel" for the shared activity "Build Dam". We guarantee some kind of convergence between spaces, but it may still require some negotiations between participants of the collaboration.

In more complex activity types several conflicts may occur. For example, let's assume the activity type of the activity "Build Dam" is extended by adding two further states "Complete Failure" and "Partial Failure" after the state "Fail". This means there can be a conflict, when the activity is changed from "Execute" to "Finish" by the fire fighter commander in the field and from "Execute" to "Fail" by the fire fighter commander in the command center. Then, the military commander changes it from "Fail" into "Partial Failure" and the fire fighter commander in the command center changes into "Complete Failure". There are now two conflicts. The algorithm can be extended to resolve several conflicts by applying it to all conflicts and by removing state changes causing the conflicts from the history (cf. for more details [7]).

Of course, the algorithm is not about handling "wrong" states. Although the states are conflicting, each party (fire fighter commander in the command center, fire fighter commander in the field or military commander) have legitimate

reasons for changing the activity states. The main goal of the algorithm is to converge to a common view based on strategic direction and defined governance roles.

D. Detecting and Handling Conflicts of Shared Activities with Dependencies

A second type of conflict can occur when two shared activities, connected via one dependency, change their state, leading to the case where the same dependency in different workspaces is in different states (e.g. in one "Neutral" and the other "Violated"). This conflict is different from the previous one and the situation causing it is illustrated in the upper part of Fig.6. The military commander has shared the activity "Transport Sandbags" with the fire fighter commander in the command center. The fire fighter commander has created a dependency "overlaps" to his own activity "Build Dam". The own activity "Build Dam" has been shared with the fire fighter commander in the field. In the bottom part of the figure, we illustrate the problem as a sequence diagram. We assume that the military commander changes the activity "Transport Sandbags" to the state "Execute" and the fire fighter commander in the field changes the activity "Build Dam" to state "Execute". The fire fighter commander in the command center cannot determine the order of state changes properly, because there might be delay when transmitting the state changes. This means the state change of the military commander is received after the state change of the fire fighter commander in the field, although the military commander changed it before the fire fighter commander. This can also lead to a different temporal order of state changes in different workspaces, since each workspace can receive state changes in different orders. Then, they have a conflicting view on the situation. We need to ensure global causality eventually, so that all participants have the same view on the situation.

Definition 5 *Eventual global causality*: $a_x : s_i < a_y : s_{i+1} \rightarrow C_j(a_x : s_i) < C_j(a_y : s_{i+1}) \; \forall AW j = 1, ..n$ sharing activity x and/or y. This definition means that when state change $a_x : s_i$ happens before state change $a_y : s_{i+1}$ then this needs to be equally observed in all AWs where the shared activities are replicated.

Lamport [8] introduced the notion of virtual time in distributed systems. It is similar to the idea in Definition 5. Virtual time progresses in terms of events, i.e. time stands still when there is no event. An event in our approach is a state change. This notion allows defining of global "happen before" relationships between state changes in each workspace. Using this notion, we can also detect in the example which state change happened before the other one (cf. our activity framework for the non-distributed setting [2]). Ensuring Definition 5 means we need to find a function C for each workspace, so that it is able to order the events in the same order like the other workspaces. Vector clocks [9] address this by using a vector containing the clock (event counter) of each workspace $n : V = (c_1, .., c_n)$. Every time a workspace i propagates a state change s to the other workspaces, it increases its counter of the vector clock (i.e.

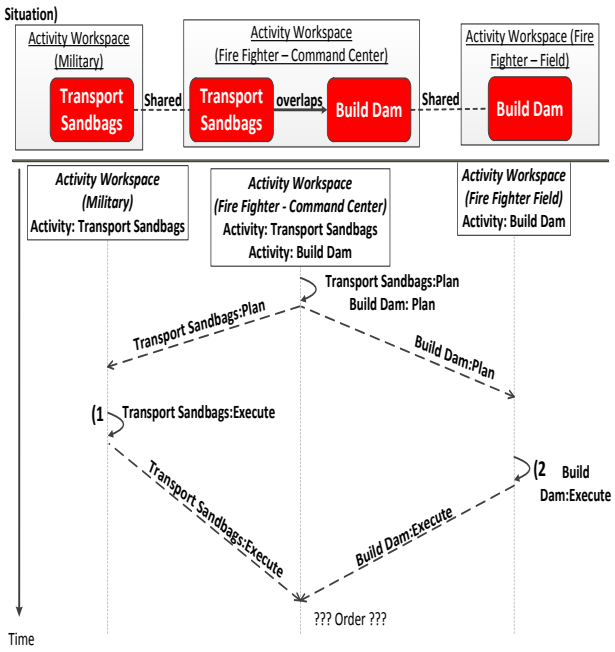

Fig. 6. Example for a situation that can cause a conflicting view on causality

the i-th item of the vector): $V[i] = V[i] + 1$. It attaches its vector clock V to the state change. A workspace receiving a state change can now put them in an order by comparing the vectors of different state changes using the following clock function C: s_i (with clock vector V_x) is partially ordered before s_j (with clock vector V_y), if : $V_x[k] \leq V_y[k] \forall k$ (otherwise they are simultaneous). It is always possible to create this partial order (cf. [9] for proofs of these concepts). The ordered state changes can be inputed into the state machine representing the dependency (cf. [2]) to detect if a dependency is violated or not and since it is the same order it will always be the same result in all workspaces.

Although the vector clock approach seems to be suitable for our purposes, it has one drawback : not everything is shared with everybody. This may lead to a situation where it is not possible to establish causality since some of the workspaces do not know about each other. For example, let us assume that in the situation illustrated in Fig. 6 the military commander in his workspace changes the activity "Transport Sandbags" into state "Execute" and propagates the state change (with Vector clock $V_{Military} = ((1, "Military"), (0, "FireFighterCommandCenter")))$ to the workspace of the fire fighter commander in the command center (illustrated in the upper part of the figure). The fire fighter commander in the field changes the activity "Build Dam" into state "Execute" and propagates the state change to the workspace of the fire fighter commander in the command center (with vector clock $V_{FireFighterField} = ((1, "FireFighterField"),$

$(0, \text{“}FireFighterCommandCenter\text{”}))$. The fire fighter commander in the command center is never able to establish causality in this case : the workspace of the military and the workspace of fire fighter in the field do not know their vector clocks. It would make the definition of temporal dependencies in this special case useless.

We solve this problem by introducing the following rule in our protocol: when a vector clock with a state change is received then the workspace i increases its own clock c_i and sends the updated clock vector to all workspaces it shares activities with.

The previous example can illustrate the effect of this rule. Suppose that the workspace of the fire fighter commander in the command center receives the state change from the military. It then updates its vector clock and sends it ($V_{FireFighterCommandCenter} = ((0, \text{“}FireFighterField\text{”}), (1, \text{“}FireFighterCommandCenter\text{”}))$) to the workspace of the fire fighter commander in the field as well as to the workspace of the military. When the fire fighter commander in the field changes the activity “Build Dam” to “Execute”, it propagates the state change together with the updated vector clock ($V_{FireFighterField} = ((1, \text{“}FireFighterField\text{”}), (1, \text{“}FireFighterCommandCenter\text{”}))$) to the workspace of the fire fighter commander in the command center. The workspace of the fire fighter commander in the command center is now able to establish causality according to Definition 5.

Approaches in distributed systems (e.g. [10]) expect that everything is shared among everybody and avoiding this kind of problem. Our approach improves other approaches in this case. More details of our approach can be found in [7].

V. Architecture

We implemented a system supporting our model with the objective experiment it with students. We used Google Wave as the underlying framework to leverage its instant collaboration and optimistic replication mechanisms around the concept of Waves. Shared documents can be distributed between different servers of different organizations (illustrated on Fig. 7 as different Wave servers). A “Wave” can have participants from different servers. The reason for choosing a collaboration platform over a simpler platform was to show how our approach works in the context of different tools needed for disaster response management (e.g. text exchange, maps, images or videos). Furthermore, it provides the infrastructure for implementing sharing of activities. Google Wave can be extended in two different ways: “Gadget” and “Robot”. A “Gadget” can be inserted into a “Wave” and is a graphical user interface to provide additional collaboration functionality (e.g. collaborating on images or collaborative modeling). It is rendered within the Google Wave Web Client in a web browser. A “Robot” can be added as an automated participant to a “Wave” and can react on events in “Waves” and modify them. It can also create further “Waves”. Google has shut down the Google Wave service and the platform has been open-sourced [11].

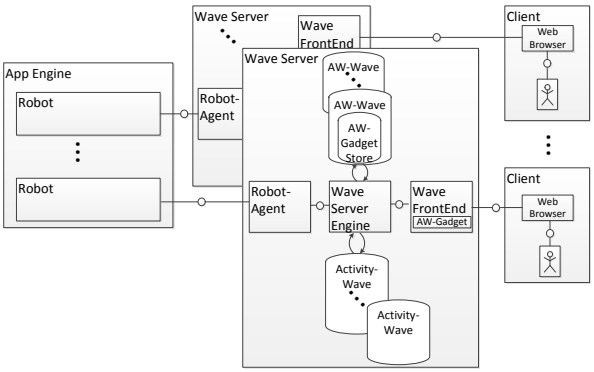

Fig. 7. Architecture of our extension

We illustrate the architecture of our extension in Fig. 7 in the context of the Wave Federation Architecture [11]. Activities and dependencies can be modeled in a special “Wave” called “AW-Wave” containing a “Gadget” providing the necessary functionality. The “Gadget” is called “AW-Gadget” and stores its data (e.g. the model) in the “AW-Wave”. The “AW-Wave” can be compared to a workspace. A robot is a distributed application on the Google App Engine or any other server. It is responsible for propagating the state changes of activities to different “AW-Waves”. Activities themselves are linked to special “Waves” called “Activity-Waves”. People can collaborate in this activity, e.g. they can insert pictures, write text or work collaboratively on a map of the situation. An activity can be shared by inviting a participant to an “Activity-Wave”. Google Wave provides already a mean for sharing and replicating activities as our approach requires it. The robot makes the shared activity available in the “AW-Waves” of participants who has been invited to the “Activity-Wave”. The activity is then shown in the “AW-Gadgets” of the “AW-Waves” of the participant and the participant can replicate it into his/her model. He can also create dependencies to his own activities. State changes can be initiated via the “AW-Waves”, where the activity is replicated. The “AW-Gadget” stores a state change as well as the vector clock in the “AW-Wave”. The robot copies the state change to all “AW-Waves” where the activity is replicated. According to the rule mentioned in the previous section, the robot copies the vector clocks to all “AW-Waves” with which activities have been shared. The “AW-Gadgets” can create the global order of state changes based on the vector clocks. They also highlight violation of dependencies to the user and they display to the user when there have been conflicting state changes according to Definition 4.

VI. Implementation

Figure 8 is a screenshot of our extension. It presents activities and dependencies in a graph. It may also provide a table view that is easier to use when someone creates a lot of activities or wants to have a quick overview of the state

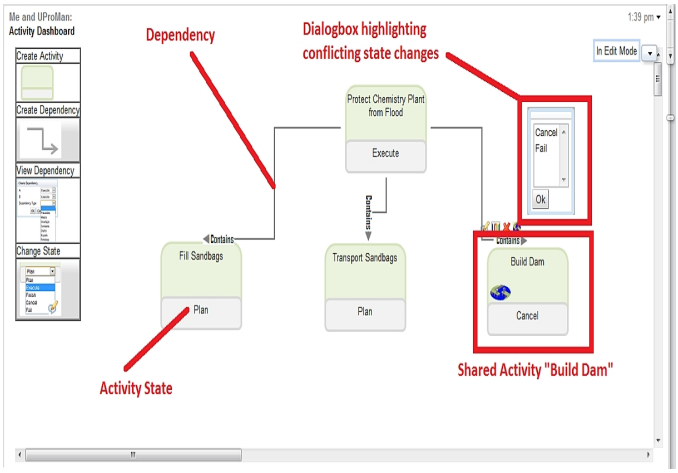

Fig. 8. Screenshot of our prototype

of activities. The figure depicts the activity workspace of the fire chief. It uses a graphical modeling notation. The fire chief sets the activity "Build Dam" conflicting to the field commander to state "Fail" and "Cancel" respectively. This is illustrated as a symbol on the activity and he has currently opened a dialog box showing the two conflicting states.

Of course, Google Wave has been shut down and our prototype is currently not available but a new initiative has been started to continue the work on the federation of Wave protocol under the Apache umbrella. The lessons that we learned from this implementation effort have not been lost under these circumstances. The Wave federated protocol for optimistic replication is effective and provided us with the right framework for the job. There is obviously a need for this kind of environment to support advanced collaborative applications in an inter-organizational setting. The Google Wave initiative was probably a bit early but we have no doubt that it will resuscitate some other day. Still, our implementation allowed us to conduct a preliminary evaluation of the proposed approach.

VII. The Challenge of the Evaluation

Designing a new model supporting new kind of user interactions between organisations is a difficult task, but it is just the beginning of the road. We also have to evaluate the system effectiveness in crisis situations. We achieved the evaluation of our approach in two steps. First we collected feedback of domain experts and second we set up a small experiment that exhibit attributes that can be compared the dynamics occurring in a crisis.

Another way to gain more insights about our solution would have been to contribute it to disaster exercises. It was too difficult and time consuming to do it. Another problem is that disaster exercises usually have a specific focus that is different from evaluating software. Although disaster managers are willing to test new software in exercises, it is not the primary objective of most of the exercises, which in turn makes it difficult to obtain valid results. We decided to use an experimental approach where we could control more of the parameters. This requires fewer resources and can be repeated more often. Additionally, with experimentation, we can focus on the tool being evaluated rather than on the situation. However, results from experiments cannot be transferred to conclusions with respect to the tool support in a disaster response. Nevertheless, they are useful to interpret the results obtained in a disaster exercise or expert interviews. Experiments have been already described for evaluating tools and concepts in the area of information systems for crisis response (e.g. [**?**]).

A. Interviews

We started validation of our approach by presenting it to four experienced disaster managers. We conducted the interviews by phone. They were recorded and transcribed. The disaster managers commented positively the general approach of activity management (cf. [2], [7]). A fire chief (Southern California) recognized the problems of traditional means for coordination :

"[..] the pile of messages in the inbox, [which contains] the reality as a situation [..] being able to put them in context and update them and coordinate them to create a common picture is the difficulty".

For example, another fire chief (Washington, DC) highlights that it allows measuring the progress of the situation and managing shifting goals (end states):

"there is a couple things [about your approach], [..] it is a good way to measure progress, and the second thing is that you recognize that the end state [goal] will change because of the dynamic situation of the incident you are involved in [..] the end state may need to be modified or you might have intermediate type of objectives".

He further says that sharing of activities (missions) is important: "[You are able to] define specific objectives that need to be accomplished in order to meet that end state [..] and then [these] objectives [are] transmitted as mission statements to the ground [..] those folks at the ground level [define] the mission, [plan] the mission , [develop] the tasks and tactics [and have to] make time-critical decisions in order to meet that mission".

Another fire chief (Southern California) confirms the previous statements: "[The approach] addresses the problem of coordination and sharing goals and objectives from one organizations to other and it is that communication [to update] information, [such as] progress as far as plan, changes made, it is that communication link that inherently seems to be the crux of all problems. [If this] sharing does not occurs [then] a lot of information stays within each independent organization [..] Without that knowledge we duplicate services, we actually implement plans that interfere with the others, goal and objectives".

Our approach can help them to have a more meaningful and accurate situation awareness on the current state of their own and others' activities. The fire chief from south of France highlights that our approach can make a difference in situation where coordination is the issue: many organizations

are involved, it is a geographically distributed situation, there needs to be time to plan and there needs to be time for communication between different actors.

Expert interviews can be used to gain a consensus about advantages and disadvantages. Based on our own experience [7], this can provide useful hints. However, we noticed also cultural differences with respect to risk attitude towards using new software. Some experts were more reluctant to accept new technologies as part of their work.

B. Design of our Experiment

Although experimental research is important, there are not many experiments described with respect to inter-organizational coordination in dynamic situations. We find some experiments about dynamic process management in the literature, but they do not address the inter-organizational dimension. Thus, we designed an experiment to assess and compare different tools for this purpose. In order for the experiment to be successful, it must demonstrate the typical coordination problems, as described before, can be reproduced. We conducted the experiment successfully three times to confirm its design. Further experiments are currently conducted to assess and compare different tools including our own prototype.

1) Details: Our experiment design is inspired from the LEGO serious play™ experiments in management science [12], [13], [14]. There, LEGO ®[1] has been used as a tool to describe and evaluate business strategies. Contrary to existing experiments in business process management (cf. [15], [16]), our experiment requires to coordinate actions in the real world and not coordinating work on a digital artifact.

During the experiment, five student teams had to coordinate the construction of a LEGO® object: architect, builder, assembler, transporter and engineer. Each team was located at a different site and could not see what the other team was doing. They had to coordinate through an assigned tool (e.g. chat tool). A LEGO® object consisted of LEGO® components, which consisted of standard LEGO® bricks. The architect team had the specification of the LEGO® object. It instructed the builder team to construct components and the transport team to transport them from the building site to the assembly site. The situation is illustrated in Fig. 9.

There, the architect instructed the assembler how to create an LEGO® object out of the components. The builder team had only the specification of the LEGO® components. The engineer team had to construct another LEGO® object, which was related to the object of the architect team. It requested LEGO® components from the builder team and assembled them itself. Since not every team knew what the other team was doing or their specifications, we expected that typical coordination problems would occur. Shifting goals can be simulated in various ways, for example, by change of specification or change of teams. We expect that this would

[1]LEGO® is a trademark of the LEGO Group of companies which does not sponsor, authorize or endorse this publication (see `http://aboutus.lego.com/corporate/fairplay.aspx`)

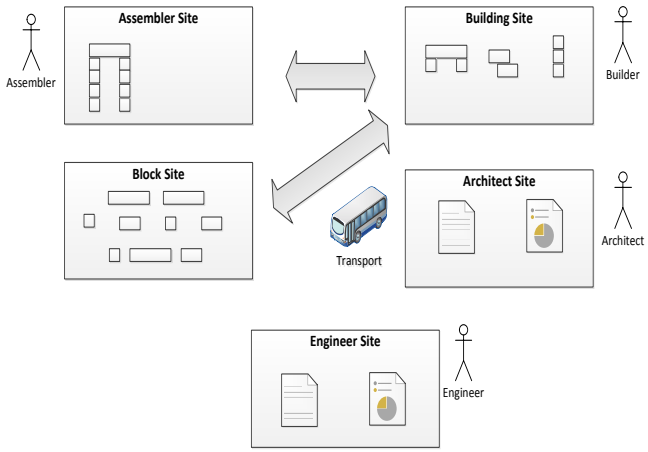

Fig. 9. Locations of the five different teams

lead to a higher probability that coordination problems will occur and think it is more closely to the disaster exercise case.

2) Outcomes: In order to assess the experiment design, we generated three different outcomes of the experiment. The first outcome was a survey conducted before and after the experiment. The survey, conducted before the experiment, assessed the expertise of the participants. Dörner showed that experienced managers have better skills to solve problems in experiments than students [17]. However, valid conclusion can be still derived from these student experiments. The survey after the experiment tried to assess which coordination problems could be related to the tools used and which ones had other reasons. For instance, we asked each team what were the main problems and what were the problems faced with each team. They had also to provide input on advantages and disadvantages they faced with their tools. We found out in the surveys that indeed problems occurred due to the tool used, but other problems had their cause in misunderstandings.

The second outcome was the data gathered from the tools used for coordination. We show in Figure 10 an example for a coordination problem caused by the chat tool used. The architect team got confused, because the builder team confirms twice that the same blue component has been completed, but it never receives information about the white component. This led to further confusion. Further conflicts have been identified in the data gathered from the tools.

The third outcome was the objects constructed by the efforts of different teams (cf. Fig. 11 for two outcomes generated in the experiment). It turns out that the objects were very close to the specification, but did not fit exactly. For instance, in one case one part of the object has to be hold by the assembler team so that it does not collapse. We could not exactly identify the root cause for this, because it could have been also a misunderstanding between architect and assembler team.

VIII. Related Work

In [2], we have compared several process management systems, addressing a disaster response scenario. These systems do not take into account an inter-organizational setting. Inter-organizational coordination of activities has mostly been addressed in the area of business process management systems. Aalst and Weske [18] propose to split a previously defined public process in several parts and let every involved party execute its part in a distributed fashion. Grefen et al. define the public process as a contract between organizations [19]. Schulz et al. [20] describe a view-based approach which is similar to the previous approach. Fdhila et al. [21] propose to execute a public process as a choreography. The approaches in [22], [23] allow defining a public process and each involved party can deviate from it. All these systems require to some extent defining a global public process or choreography before the process execution. Based on our interactions with end user, we believe this is not an acceptable assumption for autonomous organizations that may even work together for the first time. It is also not always obvious when and if they have to work together. The processes in our use case are defined top-down, but also bottom-up. They can be defined vertically and horizontally in a network of organizations. We do not require definition of a public process in advance. Furthermore, these approaches consider mostly sequential processes, which is not a realistic assumption for disaster response processes [2]. They are also limited with respect to detection of concurrent conflicts. Our approach can be compared with the unified activity management approach in [24]. However, they do not consider an inter-organizational distributed setting.

We find many approaches addressing detecting and handling of conflicts in distributed collaborative work (cf. [5]). For example, collaborative image [25] or text editing [26] with optimistic change propagation. These approaches deal with conflicts in unstructured documents (e.g. text or images) and not in structured models like our model.

Other approaches deal with the consensus problems in distributed systems (e.g. [27], [10]). The consensus problem is different from our approach because it deals with faulty processes (i.e. processes that send different values to different processes). Our problem is to find a consensus between processes what send correct values. In our approach, such a consensus is not possible, because each party has a different view on the real world (e.g. the command center has a more distant strategic view and the people in the field a more operational view) and consensus protocols cannot deal with this problem. Furthermore, the actors may have different goals and processes that interact with each other. In our scenario, the notion of faulty processes does not exist. This is why we proposed another approach based on governance roles to resolve conflicting views.

me: Build Piece (C,1)
4* Build Piece (C,1,White)
4* Build Piece (C,1,Blue)
Builder: finish(4* Build Piece (C,1,Blue))
me: Blue?
Builder: sorry
me: i asked you first the white ones :p
Builder: that's done
me: Build Piece (B,2,Grey/Red)
Builder: finished(4* Build Piece (C,1,Blue))
me: i asked you something
how is the work?
stop building
unbuild Piece (B,2,Grey/Red)
tell me when it is done
the unbuild

Fig. 10. Example for a coordination problem when using the chat tool

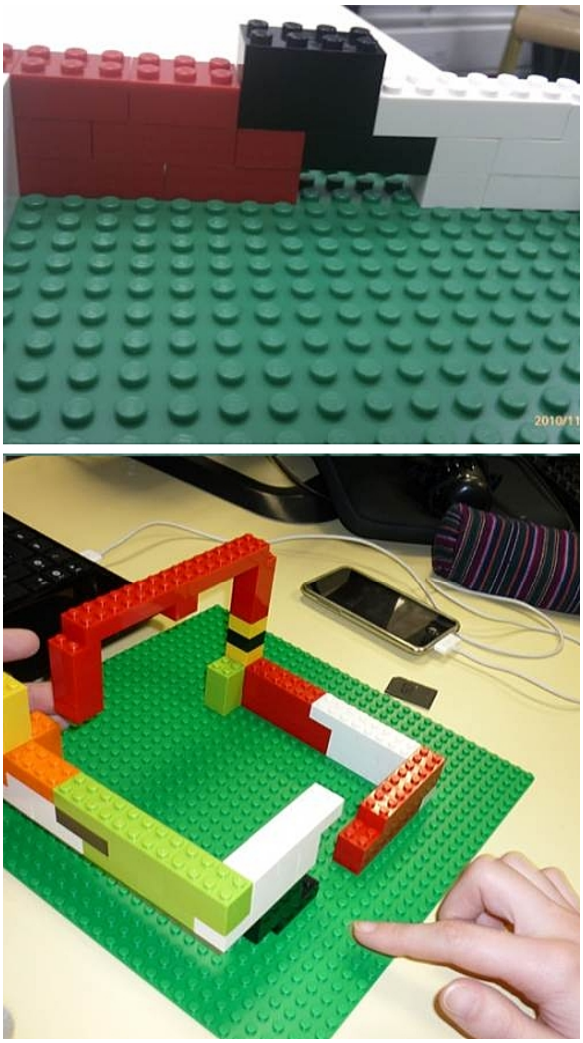

Fig. 11. Two constructed objects by different groups

IX. Conclusion

As we described it, our activity management system can support the coordination of autonomous organizations. It permits them to share selected activities with each other by replicating them in their different workspaces. Since conflict may occur, we have described how, for two different types of conflicts, the system can detect them and how it provides mechanisms for handling them. Using optimistic propagation

allows capturing the situation (activities and their state) in time and it enables to detect of views in conflict; disaster managers have acknowledge this way to proceed. Our proposition already overcomes some problems with traditional means for ad-hoc coordination (e.g. email or phone) or even more traditional BPM approaches (see related work) - they have only limited capabilities for detecting and handling concurrent conflicting views on dynamic processes which are partially known by each organization. In the future, we want to explore how to establish the social network between actors sharing activities based on pre-defined plans. This also includes the definition of governance roles. Disaster managers have commented positively our approach [28], [2]. We started first evaluations in form of experiments with students. We validated the design of the experiment by conducting it three times. Experiments provide additional value and should be conducted jointly with expert interviews or disaster exercises. We expect that our approach is applicable to other dynamic collaborative scenarios, such as organization of large events (e.g. Olympic Games) or development projects. Our concepts presented here can also be adapted to extend other constraint-based process management systems to the inter-organizational level (e.g. [29]).

X. Acknowledgements

The research was partially funded by the French Ministry of Research within the RESCUE-IT project [30]. We thank the domain experts for providing us detailed insights.

References

[1] E. Olding and C. Rozwell, "Expand your bpm horizons by exploring unstructured processes," Gartner, Tech. Rep. G00172387, 2009.

[2] J. Franke, F. Charoy, and P. El Khoury, "Framework for coordination of activities in dynamic situations," *Journal of Enterprise Information Systems*, vol. iFirst, 2012.

[3] S. Döweling, F. Probst, T. Ziegert, and K. Manske, "Soknos - an interactive visual emergency management framework," in *Workshop on Geographical Information Processing and Visual Analytics for Environmental Security*, 2009.

[4] J. F. Allen, "Maintaining knowledge about temporal intervals," *Communications of the ACM*, vol. 26, no. 11, pp. 832–843, 1983.

[5] S. Greenberg and D. Marwood, "Real time groupware as a distributed system: Concurrency control and its effect on the interface," in *ACM CSCW Conference on Computer Supported Cooperative Work*, 1994.

[6] J. Franke, F. Charoy, and C. Ulmer, "Coordination and situational awareness system for inter-organizational disaster response," in *10th IEEE International Conference on Technologies for Homeland Security*, 2010.

[7] J. Franke, "Coordination of distributed activities in dynamic situations - the case of inter-organizational crisis management," Ph.D. dissertation, École Doctorale IAEM Lorraine, Université Henri Poincaré/Université de Lorraine, Nancy, France, 2011.

[8] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[9] F. Mattern, "Virtual time and global clocks in distributed systems," in *Workshop on Parallel and Distributed Algorithms*, 1989.

[10] U. Bartlang and J. P. Müller, "Dhtflex: A flexible approach to enable efficient atomic data management tailored for structured peer-to-peer overlays," in *3rd International Conference on Internet and Web Applications and Services*, 2008.

[11] Google, "Wave in a box, http://www.waveprotocol.org, retrieved 25.08.2010."

[12] P. Bürgi, B. Victor, and J. Lentz, "Modeling how their business really works prepares managers for sudden change," *Strategy & Leadership*, vol. 32, no. 2, pp. 28–35, 2004.

[13] M. Statler and D. Oliver, *The Oxford Handbook on Organizational Decision-Making*. Oxford University Press, 2008, ch. Facilitating Serious Play, pp. 475–494.

[14] K.-P. Schulz and S. Geithner, "The development of shared understandings and innovation through metaphorical methods such as lego serious play (tm)," in *International Conference on Organizational Learning, Knowledge and Capabilities*, Hull, UK, 2011.

[15] B. Weber, H. A. Reijers, S. Zugal, and W. Wild, "The declarative approach to business process execution: An empirical test," in *21st International Conference on Advanced Information Systems*, Amsterdam, The Netherlands, 2009.

[16] B. Mutschler, B. Weber, and M. Reichert, "Workflow management versus case handling: Results from a controlled software experiment," in *23rd Annual ACM Symposium on Applied Computing*, Fortaleza, Ceará, Brazil, 2008.

[17] D. Dörner, *The Logic of Failure: Recognizing And Avoiding Error in Complex Situations*. Basic Books, 1997.

[18] W. van der Aalst and M. Weske, "The p2p approach to interorganizational workflows," in *13th International Conference on Advanced Information Systems*, 2001.

[19] P. Grefen, K. Aberer, Y. Hoffner, and H. Ludwig, "Cross-flow: Cross-organizational workflow management in dynamic virtual enterprises," ESPRIT project No, 28635, European Comission, Tech. Rep., 2000.

[20] K. A. Schulz and M. E. Orlowska, "Facilitating cross-organisational workflows with a workflow view approach," *Data & Knowledge Engineering*, vol. 51, pp. 109–148, 2004.

[21] W. Fdhila, U. Yildiz, and C. Godart, "A flexible approach for automatic process decentralization using dependency tables," in *7th IEEE International Conference on Web Services*, 2009.

[22] J. C. Grundy and J. G. Hosking, "Serendipity: integrated environment support for process modelling, enactment and work coordination," *Automated Software Engineering*, vol. 5, no. 1, pp. 27–60, 1998.

[23] S. Rinderle, A. Wombacher, and M. Reichert, "Evolution of process choreographies in dychor," in *On the Move Conferences*, 2006.

[24] B. L. Harrison, A. Cozzi, and T. P. Moran, "Roles and relationships for unified activity management," in *International ACM SIGGROUP Conference on Supporting Group Work*, 2005.

[25] C. Sun and D. Chen, "Consistency maintenance in real-time collaborative graphics editing systems," *ACM Transactions on Computer-Human Interaction*, vol. 9, no. 1, pp. 1–41, 2002.

[26] A. Imine, M. Rusinowitch, G. Oster, and P. Molli, "Formal design and verification of operational transformation algorithms for copies convergence," *Theoretical Computer Science*, vol. 351, no. 2, pp. 167–183, 2006.

[27] L. Lamport, "The part-time parliament," *ACM Transactions on Computer Systems*, vol. 16, no. 2, pp. 133–169, 1998.

[28] J. Franke, F. Charoy, and C. Ulmer, "A model for temporal coordination of disaster response activities," in *7th International Conference on Information Systems for Crisis Response and Management*, 2010.

[29] W. van der Aalst and M. Pesic, "Decserflow: Towards a truly declarative service flow language," in *Web Services and Formal Methods*, 2006.

[30] A. Schaad, C. Ulmer, and L. Gomez, "Rescueit - sécurisation d'une chaine logistique internationale," in *Workshop Interdisciplinaire sur la Sécurité Globale*, 2010.